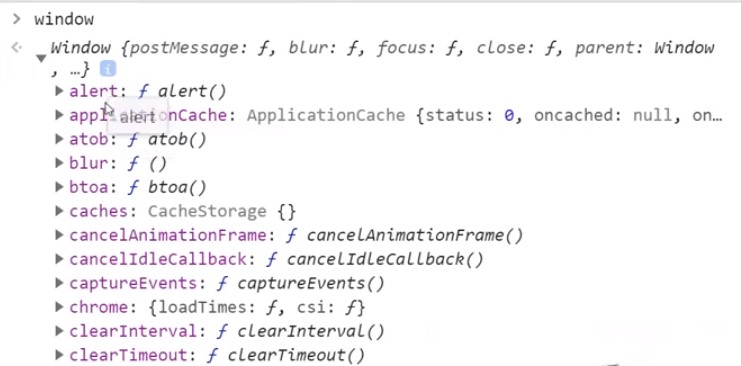
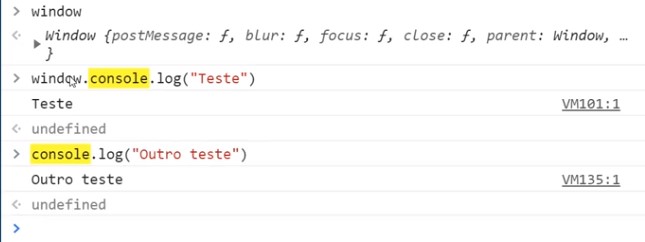
**JAVASCRIPT INTERMEDIÁRIO (IMPORTANTE)**

➤ **WINDOW:** É o objeto mais importante de todos, ele é um objeto padrão que existe em toda página da internet, todo navegador gera o objeto WINDOW e se ele for aberto veremos quantos métodos e propriedades que ele tem, são vários, mas o mais importante deles é o console! O método console que estamos utilizando até então é um método do objeto WINDOW.



Como a internet está em constante evolução, não precisamos invocar o WINDOW para convocar a sua propriedade console, podemos fazer isso direto, mas as duas formas funcionam:



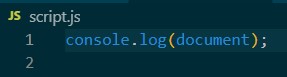
Das duas maneiras eu chamo o Objeto Window, invocando a sua propriedade Console, mas em um eu menciono o Window e no outro NÃO!

➤ **DOCUMENT:** É uma das propriedades mais importantes do **OBJETO WINDOW**, através do document acessamos o nosso arquivo HTML, vamos ver isso na prática:



A propriedade do OBJETO WINDOW “document” ele vai conter TODO O NOSSO HTML!

assim como o console, podemos acessar digitando apenas “document” no console da página, ou pelo script.js:



E como o “document” acessa o nosso HTML, podemos brincar à vontade com o HTML, através do JAVASCRIPT. Você pode acessar todo o HTML digitando apenas document, ou acessar partes específicas, especificando qual você quer:

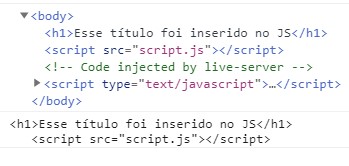
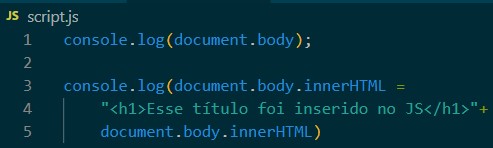


Indo mais fundo, você pode escolher apenas uma parte dentro do BODY e INCLUSIVE mudar a sua função:



Fazendo isso, ele mudaria TODO o “document.body.innerHTML”, para preservar o que tinha antes dentro do “innerHTML” nós concatenamos/adicionamos: +document.body.innerHTML no final da inserção do “<h1>Esse título foi inserido pelo JS</h1>” e assim adicionamos o primeiro e mantivemos o segundo.

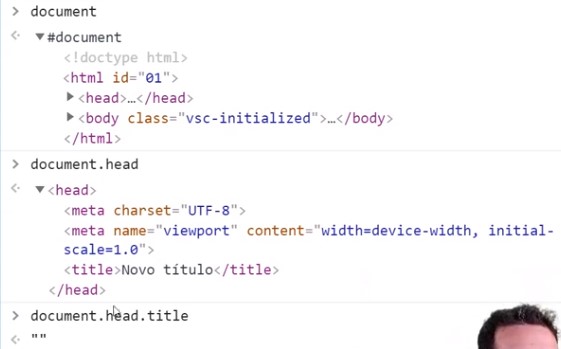
Lembrando que isso pode ser feito pelo próprio navegador, no console, como mostrado acima, ou pelo VS CODE:



Então imprimindo novamente o “document” veremos que lá agora tem um texto em <h1></h1> que antes não existia:



➤ **ACESSANDO ELEMENTOS DO HTML com o JS:** Como visto anteriormente, podemos acessar os elementos do HTML através do “document” isso funciona com o body e com o head, e com alguns outros, mas veremos que alguns elementos como o title, não são acessados dessa forma:

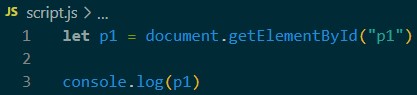
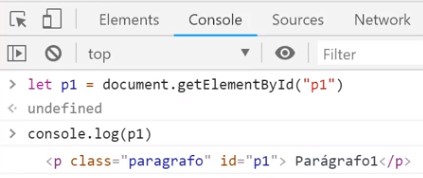


Quando você tenta acessar o title, ele não mostra nada, como se estivesse inacessível, então como podemos acessar o title pelo JS?

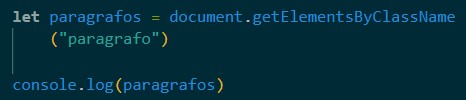
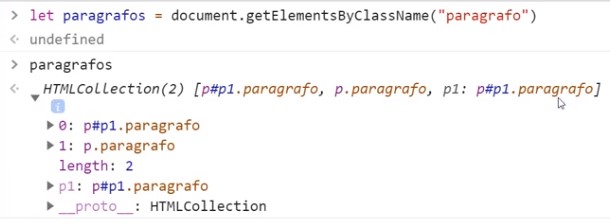
Bom, suponhamos que originalmente, sem mudar nada, nós temos:



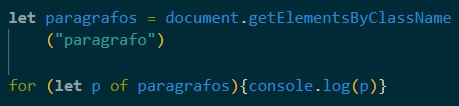
ACESSAREMOS as coisas com o JS através do: “**getElementById”** mas obviamente, o elemento PRECISARÁ ter um ID, como veremos a seguir:



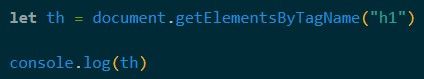
Da mesma forma que podemos puxar pelo ID, também podemos puxar pela classe:



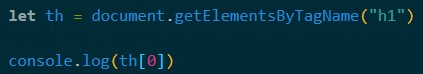
Ou pedindo para ver o que tem exatamente dentro dos parágrafos:



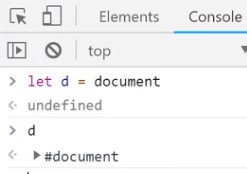
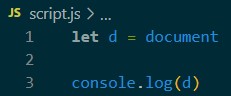
E qual a maneira de pegar TODOS os elementos? Ou partes dele?



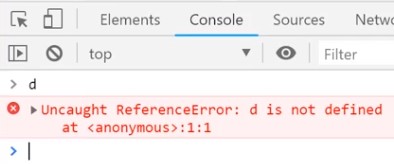
**OU**



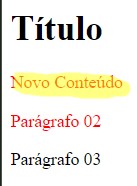
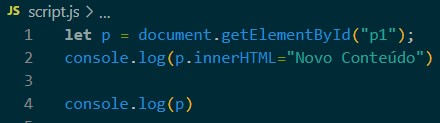
➤ **MODIFICANDO ELEMENTOS:** Nós podemos criar uma variável “d” por exemplo, e atribuir a ela, a propriedade document, e essa variável funcionará como se fosse o próprio “document”



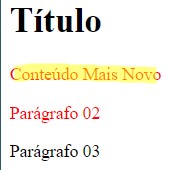
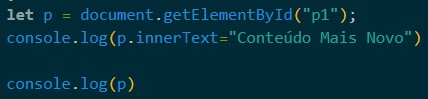
Mas se imprimirmos isso apenas pela própria página, caso você atualize a página e queira imprimir novamente a variável d, ela não vai mais estar atribuída ao “document” e ocorrerá um erro.



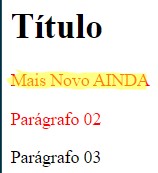
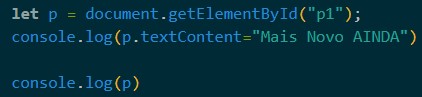
E para modificarmos o conteúdo, podemos fazer:



**OU**

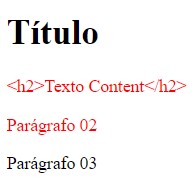
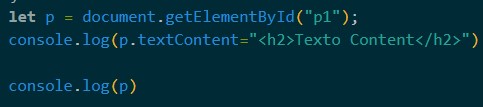
****

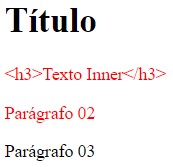
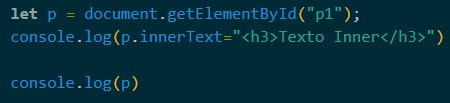
**OU**

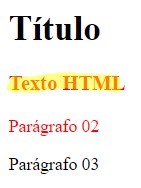
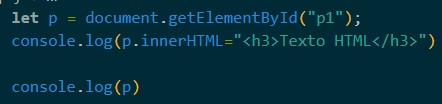
****

E QUAL A DIFERENÇA DOS TRÊS, se todos aparentemente fazem a mesma coisa?

➜ **“innerHTML”:** Vai receber um HTML, e os outros dois não. Exemplo:

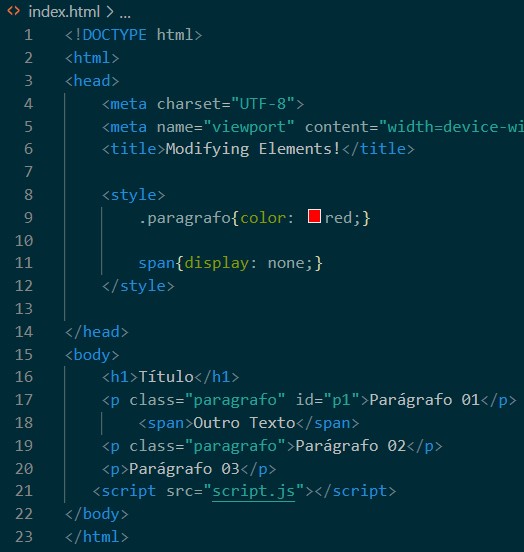




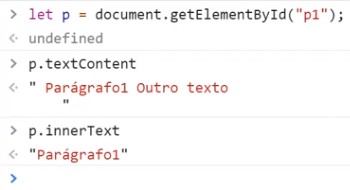


Então o “innerHTML” colocou o <h3></h3> DE FATO no texto, deixando ele do tamanho escolhido.

E o innerText do textContent? Qual a diferença entre eles? Vamos supor que temos:



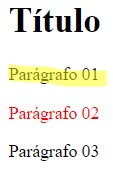
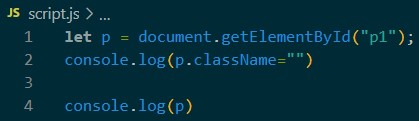
Ou seja, apesar de termos um span “Outro Texto” que apareceria do lado de “Parágrafo 01”, como colocamos “style no span de: display: none” esse “Outro Texto” some e não aparece, mesmo estando lá.



Então essa é a diferença básica entre eles, o “innerText” vai me mostrar só o que está aparecendo na tela, já o “textContent” vai mostrar todos os textos que está aparecendo no nosso HTML.

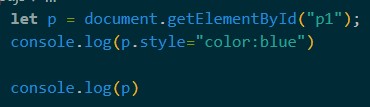
➜ **“className”:** Com ele podemos modificar os atributos de um elemento, que no caso do nosso p, que estava em vermelho, quando mudamos os atributos para nada, ele volta a ser preto:

ANTES

Depois

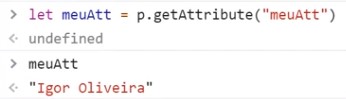
Ou seja, o parágrafo 01 perdeu a sua classe, perdendo também o seu ESTILO!

Podemos mudar através do JS o estilo das coisas também:



Deixando o meu p da cor azul.

**OBS: “Adicionando Atributos”:** Posso colocar mais atributos no meu p, e depois mudar caso eu queira, nesse caso eu precisarei atribuir o nome para ele, e depois sim mandar buscar o meuAtt:

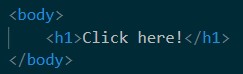


Posso também depois mudar o atributo “Igor Oliveira” para outro atributo:



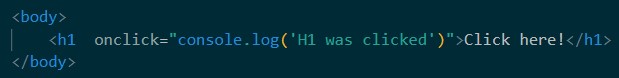
➤ **EVENTOS - DOM Events:** São eventos do “document object model” e a partir dessa aula, entenderemos como de fato vamos integrar o html com o javascript.

Primeiramente criaremos um texto normal no html:



E nos atributos do HTML, utilizaremos um atributo chamado ONCLICK

➜ **“onclick”:** Existem vários deles, são atributos de EVENTOS, ou seja, são atributos que disparam eventos e com ele eu consigo executar um código JS quando o EVENTO de fato acontecer. Nesse caso, colocaremos esse atributo no <h1></h1>, e esse **onclick** faz com que algo, quando for clicado, vai ser contabilizado.

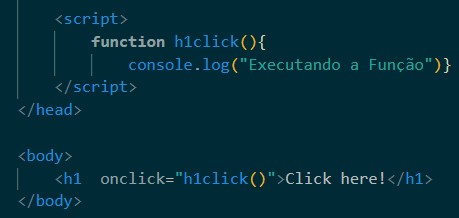




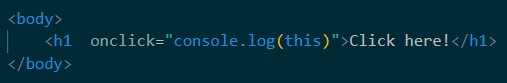
**OBS: “LEMBRANDO”:** Como eu usei aspas duplas lá fora, eu obrigatoriamente precisarei usar aspas simples dentro, para o código não entender que eu estou fechando as aspas.

Ou seja, toda vez que você clicar no click aqui no navegador, o console vai contabilizar.

Isso abre pra gente toda uma nova gama de possibilidades, dentro do ONCLICK você pode colocar qualquer código JAVASCRIPT, ou seja, em vez de console.log poderia ser uma função, por exemplo:



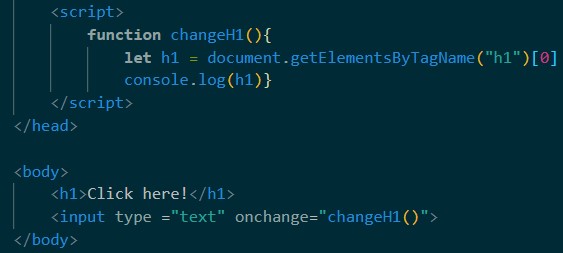
Já no caso do this, quando você clicar no “Click Here” o “this faz mostrar o próprio elemento:



Ou podemos também mudar a cor do “click here” ao clicar nele:



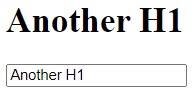
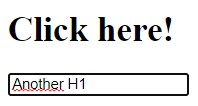
➜ **“onchange”:** É um outro evento, que faz:



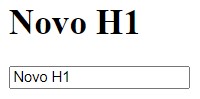
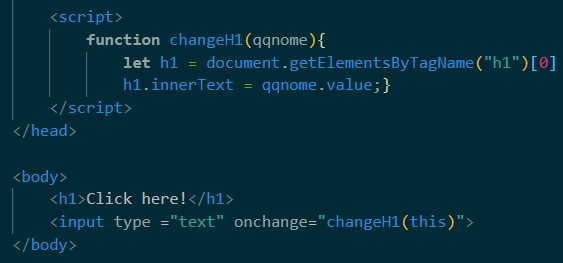


Sempre que você digita na janela, e clica fora dela, ele imprime o h1

Ou podemos também modificar o título do <h1></h1> pelo que foi digitado na janela da seguinte forma:



**OU**

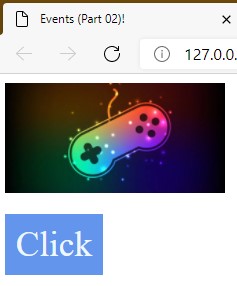


Como último exemplo, veremos como esconder o seu H1:

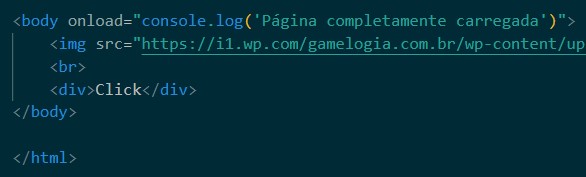


Quando você clica no botão “Click”, o texto H1, desaparece!

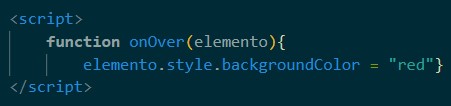
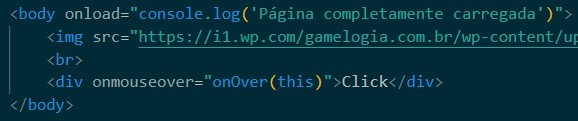
➤ **EVENTOS (Parte 02):** Aqui mostraremos mais eventos disponíveis: ANTES:



➜ **“onload”:** Quando a página estiver completamente carregada, depois de ter baixado todas as imagens e etc, aparecerá esse aviso no console, ou seja, é um evento que vai ser executado quando a página terminar de carregar.

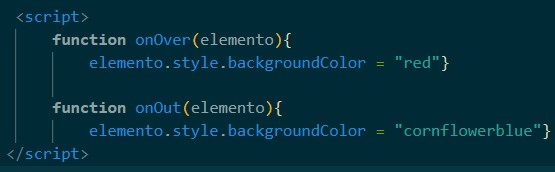
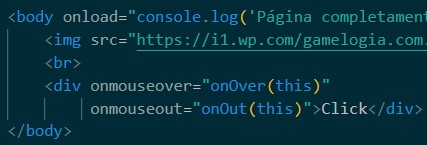
➜ **“onmouseover”:** Esse evento faz algo mudar quando colocamos o mouse por cima do elemento escolhido, nesse caso, ao passar o mouse por cima da DIV Click, ela mudou o fundo para vermelho.

**OBS: “LEMBRANDO”:** No CSS, para mudar o fundo de alguma coisa, digitamos: “background-color”, já no JS, para conseguir mudar o fundo, digitamos: “backgroundColor”.

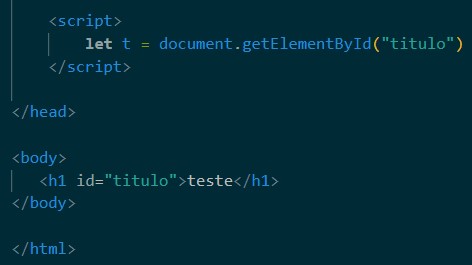
Mas e caso eu quisesse que ao tirar o mouse, o botão voltasse para a cor original?

➜ **“onmouseout”:** Faz o que foi feito ser desfeito ao tirar o mouse:

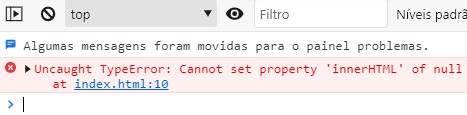
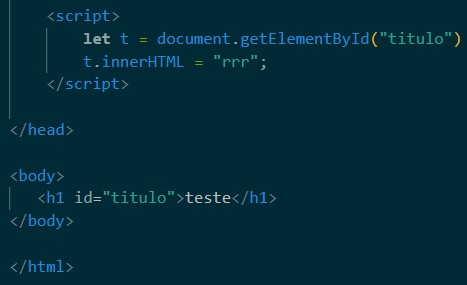




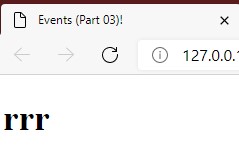
➤ **EVENTOS (Parte 03):** Continuando com eventos:



Já sabemos que para mudar o que está escrito dentro do <h1></h1>, basta colocar um innerHTML, mas o que acontece se estiver no lugar errado?

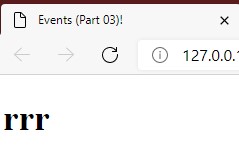


Dá erro, pois a ordem é muito importante! Para funcionar, o nosso script deveria estar depois do <h1>, dentro do <body>, porque o título precisa ser executado primeiro, para depois ser mudado, então a ORDEM É MUITO IMPORTANTE.



Mesmo se o <script> estivesse no <body>, mas ANTES do <h1> ELE NÃO FUNCIONARIA, tem que ser depois do <h1>. Como resolver esse problema, já que o <script> deve ficar dentro do <head>?

➜ **“onload”:** Podemos usar o “onload” para ajudar nesse problema, executando previamente o <h1> lá no <script>, dentro da <head>.



Falamos tudo isso, para exemplificar que podemos fazer a mesma coisa com o “onclick” podemos mudar ele a partir do Javascript, dentro do HTML, e não no próprio HTML:

➜ **“onclick”:** Quando eu clico no elemento, ele vai mudar conforme solicitado:



Mas existe uma segunda maneira para poder fazer o “onclick” funcionar diretamente do <script> do JS:



Poderia ser qualquer evento, dentro desse “addEventListener” como por exemplo o “onmouseover”:

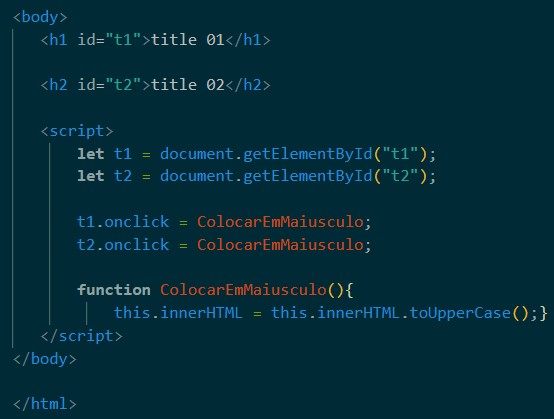


Apenas lembrando de tirar o “on”, deixando apenas “mouseover” e tudo minúsculo, senão não funciona.

Colocando tudo, inclusive o “mouseout”:

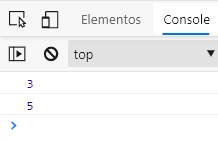
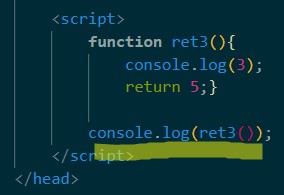
  

➤**WHAT IS “THIS”?:** This significa ISSO, quando falamos isso, precisamos entender qual o contexto para identificar o que ISSO significa, então o mesmo ocorre com o THIS, this vai seguir o contexto do que está acontecendo no nosso código.



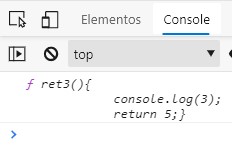
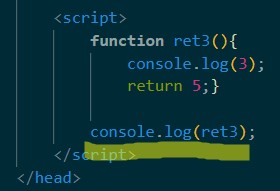
➤**PARÊNTESES:** E aí? Temos que usar ou não parênteses quando estamos falando de uma função? Quando devemos usar os parênteses e quando não devemos usar, para a função funcionar?

Tudo depende do que você quer! Se você quer executar a função, e que o navegador mostre o RESULTADO, você vai usar os parênteses:



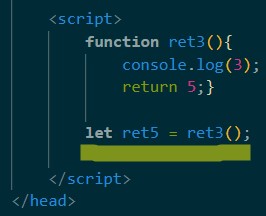
Como colocamos o parênteses depois do ret3, ele vai imprimir o 3 e logo depois retornar o número 5, que é o resultado da função.

Agora se não quisermos executar a função, e apenas ver apenas como a função está escrita, você não coloca o parênteses, deixa apenas “ret3”:

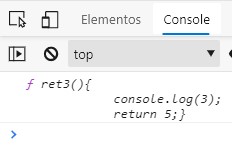


OU SEJA, quando USAMOS os parênteses, nós estamos executando aquela função e quando NÃO USAMOS parênteses, estamos apenas RETORNANDO a função, imprimindo ela novamente.

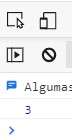
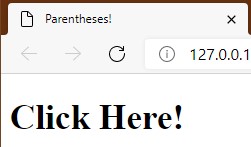
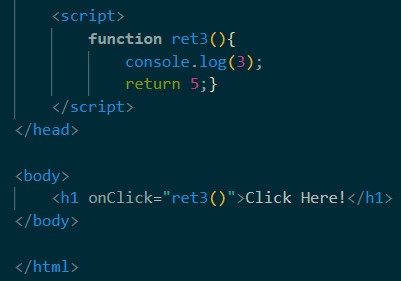
Quando atribuímos “ret3” a outro elemento, já mandando imprimir o ret3, quando digitamos esse outro elemento sem parênteses, ele já vem impresso, mesmo sem parênteses:



Pois o ret3 está retornando 5 e colocando dentro do ret5, caso o “ret3” não tivesse os parênteses lá no let, quando mandasse imprimir o ret5, ele voltaria a própria função ret3.



➜ **“HTML”:** Aqui, como eu não estou no JS eu vou ter que passar os parênteses, porque aqui eu estou no HTML, só é javascript, porque eu estou dentro da tag <script>, como aqui eu estou no HTML, para ele entender que isso não é um TEXTO “ret3” e sim uma FUNÇÃO, é NECESSÁRIO os parênteses, mas como eu estou no HTML, mais uma vez, ele vai se comportar diferente, ele não vai tentar executar assim que ele ler os parênteses, ele vai simplesmente entender que eu estou falando: “quando tiver o “onClick” eu vou precisar executar o “ret3”. Então quando clica no “Click Here!”, ele executa o ret3(3):



➤**COMPARANDO MANEIRAS DE CRIAR UM EVENTO:** Faremos uma comparação com um evento criado no HTML com “onClick” com um evento criado no “addEventListener” pelo JS.

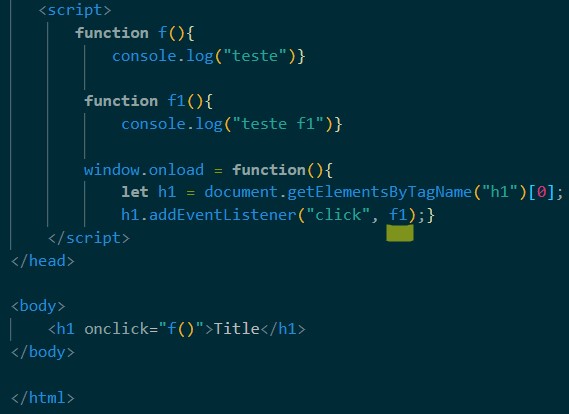
➜ **“HTML”:** Quando criamos o Evento pelo HTML, precisamos obrigatoriamente, abrir e fechar os parênteses, para o HTML entender que é UMA FUNÇÃO!

Quando Clicka: 

➜ **“JS”:** Pelo JS, quando você cria um outro evento no <script> mesmo antes de clicar no “Title”, ele já imprime, pois o JS é APRESSADO:

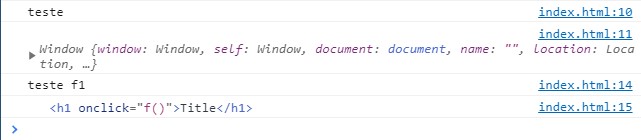


Como fazer para ele imprimir apenas depois de clicar? BASTA TIRAR OS PARÊNTESES DO “F1”:

Quando clicar: 

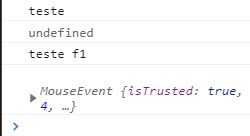
Ele vai imprimir os dois testes, pois os dois ativam quando você clica no <h1>

➜ **“This”:** E se usarmos o this, depois de cada um, o que vai acontecer com eles?

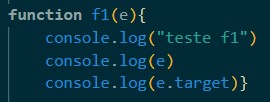


No HTML, o this vai ser o objeto WINDOW, já no JS o this vai ser o ELEMENTO.

➜ **“Nomeando a FUNCTION”:** O que acontece quando você dá valor à function?



O primeiro teste, sai indefinido, pois lá no HTML, o “f()” está vazio. Já no teste f1, ele mostra o evento “onClick”, o que nos permite fazer:

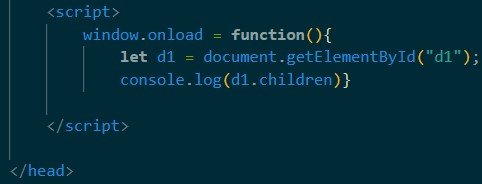
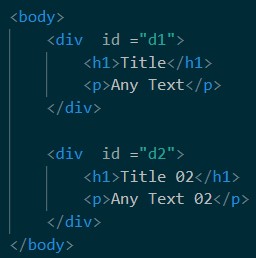


Além de imprimir o Evento do “onClick” ele vai imprimir a função do <h1>

E como fazer para a função “f” também imprimir alguma coisa? Basta colocar “this” dentro dos parênteses do “f” no HTML:



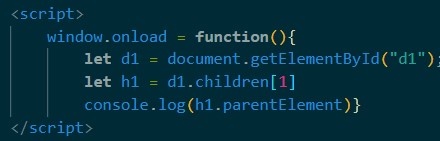
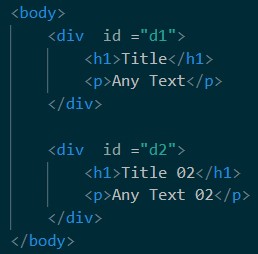
➤**ELEMENTOS PAI E FILHOS:** Existem elementos pais e elementos que descendem desses pais, que são os FILHOS, para descobrir os elementos filhos, podemos fazer por exemplo:



E com isso podemos escolher apenas uma das linhas para poder ver direito o que tem dentro:



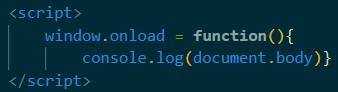
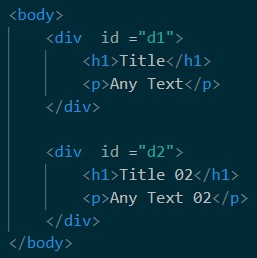
Perfeito, descobrimos como achar os filhos de um elemento, mas e como descobrir qual é o PAI de certo elemento? Simples:



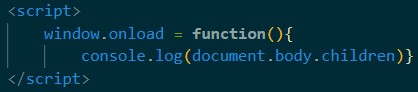
O pai do elemento é a <DIV>!

**OUTRAS FORMAS:**

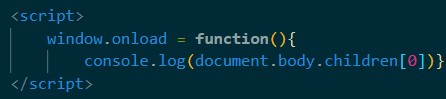
➜ **“document.body”:** Aqui você vai ver todos os elementos dentro do body:



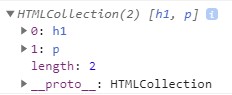
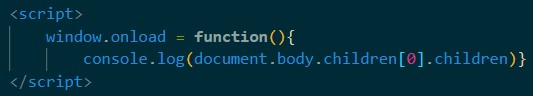
➜ **“document.body.children”:** Aqui mostra a lines dentro do body:



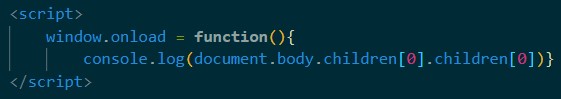
➜ **“document.body.children[0]”:** Aqui você escolhe uma das lines, e vê o que tem nela:



➜ **“document.body.children[0].children”:** E por último, você vê os filhos dessa line:

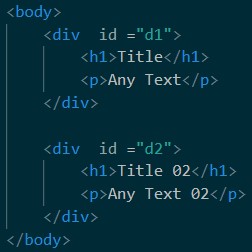


➜ **“document.body.children[0].children[0]”:** Inclusive pegar só um dos childrens para olhar o código:

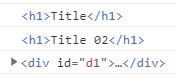


**OU SEJA:**

Dependendo do que estamos criando, pode surgir a necessidade de acessar o pai ou o filho de algum determinado elemento, e essas propriedades são importantes para isso, podemos inclusive dar outro exemplo utilizando o evento “CLICK”:

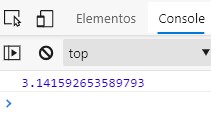
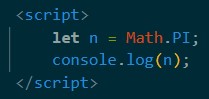


E quando você clica no título 1:

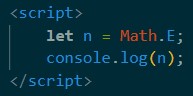
 ou no título 2: 

➤**BIBLIOTECA MATH:** O objeto math tem diversas propriedades e módulos que nós podemos utilizar para facilitar na execução de alguns cálculos. São muitas opções. como por exemplo:

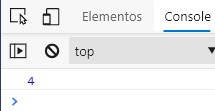
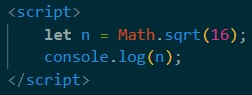
➜ **“PI”:** Mostra o PI



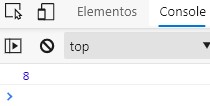
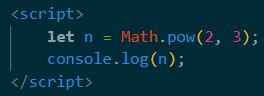
➜ **“E”:** Serve para fazer cálculos matemáticos diversos:



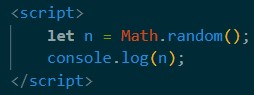
➜ **“sqrt”:** Podemos calcular a raiz quadrada:



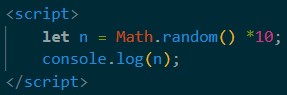
➜ **“Potência (pow)”:** Podemos calcular também a potência, aqui precisamos de dois valores, o primeiro vai ser o valor base e o segundo vai ser o valor do exponencial:



➜ **“Random”:** Gera um número aleatório de 0 até 0.9999999999999…, até QUASE 1.

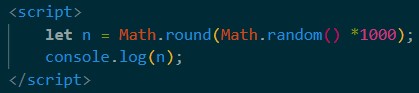


Se quisermos mais que isso, basta multiplicar, e agora estamos criando um número aleatório entre 0 e 10:

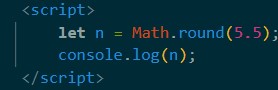
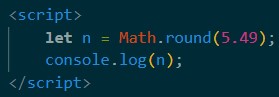


Mas e se eu quiser um número inteiro? Como eu faço?

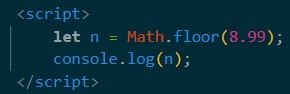
**OBS: ”Math.round”:** Faz criar um número inteiro do que eu quiser:



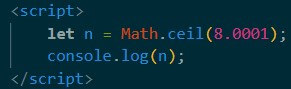
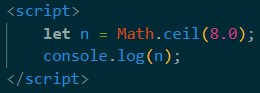
Arredondando: até 0.49 ele arredonda para baixo, de 0.5+ ele arredonda para cima.



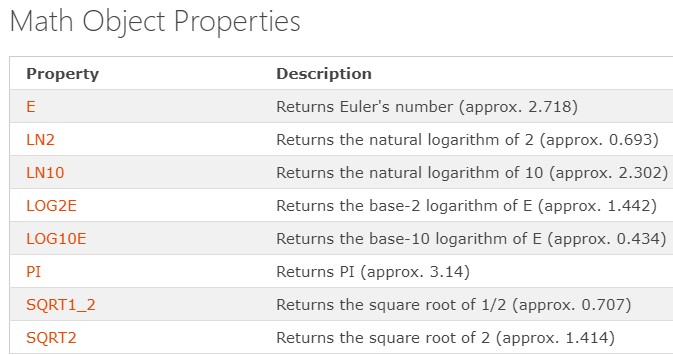
E se eu quiser que ele SEMPRE arredonde para baixo? **”Math.floor”**

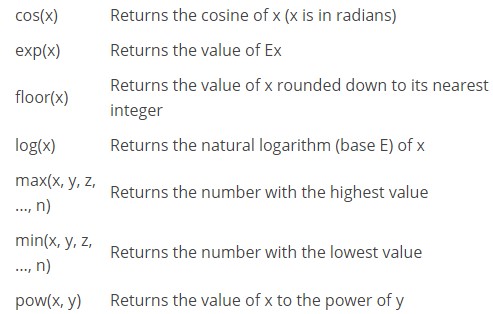
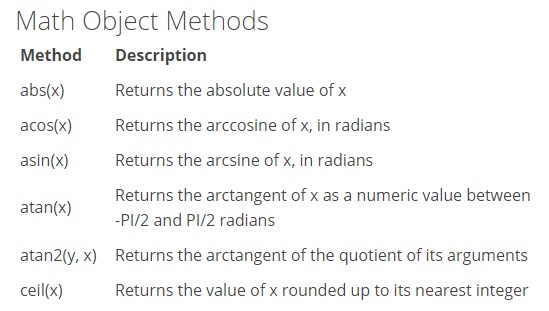


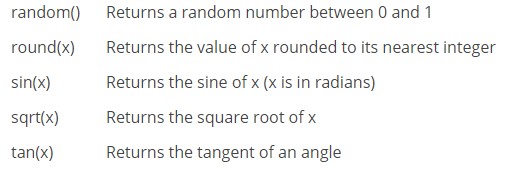
Ou sempre arredonde para CIMA? **”Math.ceil”**



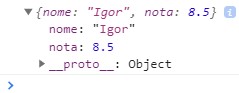
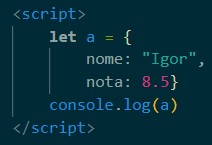
➜ **“LISTA”:** Abaixo deixaremos uma lista com mais coisas que podemos fazer com math:







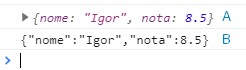
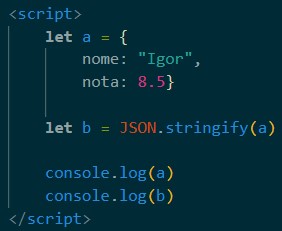
➤**JSON (JavaScript Object Notation):** Vamos supor um objeto:



Temos uma função chamada JSON, o que é ele?

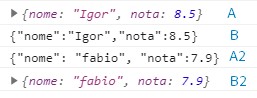
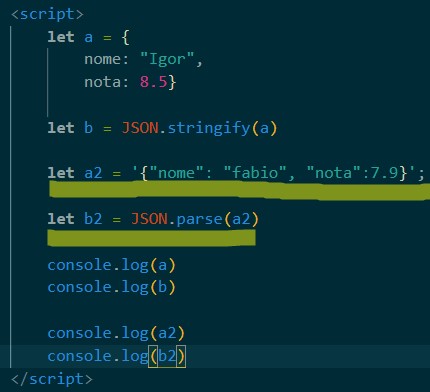
É a representação de um objeto, ou seja, é a notação de um objeto Javascript, é uma forma da gente representar um objeto JS. Existem 2 métodos para isso:

➜ **“JSON.stringify”:**  Ele pega um objeto JSON e transforma em uma STRING;



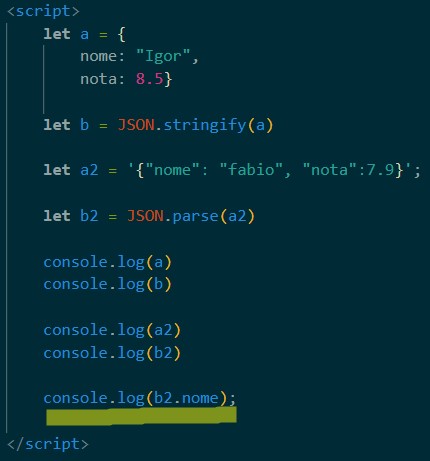
O “a” é um objeto normal, já o “b” virou uma “STRING” com a ajuda do JSON.

➜ **“JSON.parse”:**  Faz o contrário, ele pega uma STRING e tenta transformar em um objeto JSON!



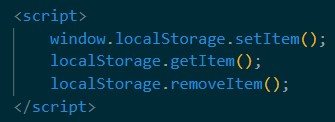
O Parse só vai identificar a “String”(A2) e a transformará em objeto (B2) se você fizer o código correto, então o código A2 tem que ter “ASPAS SIMPLES” fora do colchete, e “ASPAS DUPLAS” dentro de cada palavra dentro do colchete, números não precisam ficar dentro de aspas.

**OBS: ”Único objeto”:** Você não consegue pegar um único elemento da “STRING” pois ela é algo inteiriço, mas você pode selecionar apenas um elemento de dentro do “OBJETO”:

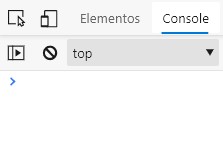
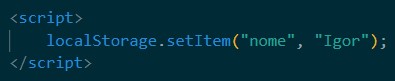


➤**LOCAL STORAGE:** Até então, tudo que criássemos no JS, se perdia quando déssemos um update na página, ou fechássemos o navegador, tudo isso fazia com que as modificações que a gente fez, se perdessem. Mas existe uma maneira de armazenarmos essas modificações e podermos buscar mesmo que fecharmos o navegador e voltarmos apenas mais tarde.

O nosso objeto window tem uma propriedade chamada: LOCAL STORAGE, e ele tem 3 itens interessantes: OBS: você pode colocar window na frente, ou omitir o window, tanto faz.

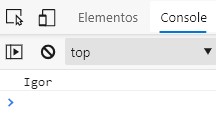


➜ **“setItem”:**  Ele vai gravar um dado na memória/arquivos do navegador e poderemos utilizar esse dado posteriormente.

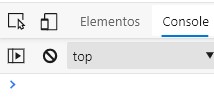
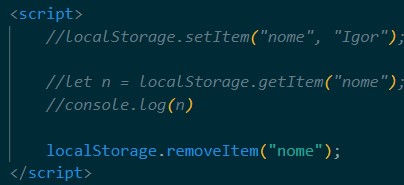


Aparentemente está vazio, mas o setItem é assim mesmo, ele não mostra nada, APENAS GRAVA!

➜ **“getItem”:**  E como eu pego essas memórias gravadas depois? COM “GETITEM”. O “getItem vai lá na memória do navegador e ele pega esse dado que estamos querendo que está lá.

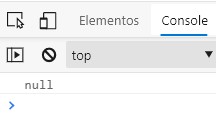
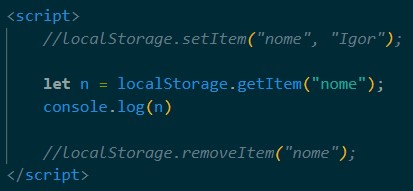


➜ **“removeItem”:**  Vai na memória do navegador e apaga o dado de lá, e passamos a não ter mais acesso a ela.



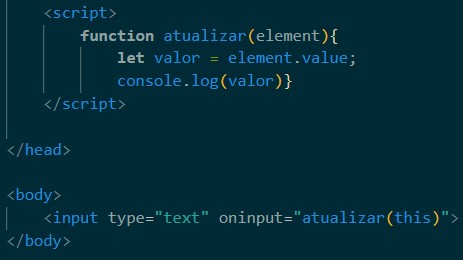
Também não aparece nada, pois só apaga os dados gravados escolhidos.

Vamos ver se o remove funcionou?

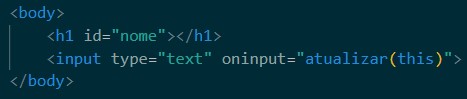


SIM! ele funcionou.

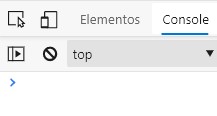
➜ **EXEMPLO:**  Vamos fazer outro exemplo, onde nós temos uma janela de digitação, e queremos gravar tudo que for digitado lá… inicialmente não vai gravar nada:



Vamos fazer com que o nome digitado na janela, seja inserido em um Título, e que fique gravado lá, até você inserir um novo nome:

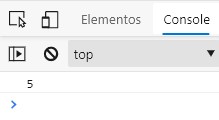
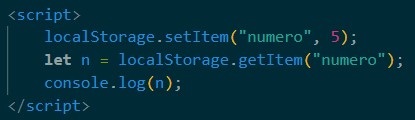


Com esse código, quando digitarmos um nome, ficará gravado, e quando digitarmos outro, substituirá:

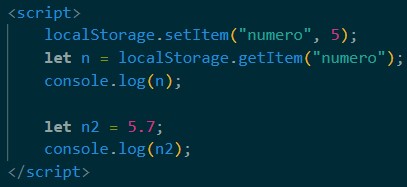


Essa é uma maneira de fazer PERSISTIR os dados, ou seja, PERSISTÊNCIA DE DADOS é uma maneira de você gravar os dados em algum lugar para depois ser acessado e não perdidos, quando fecharmos o navegador.

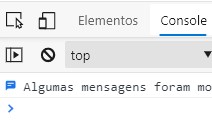
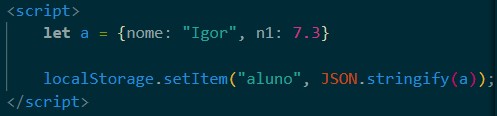
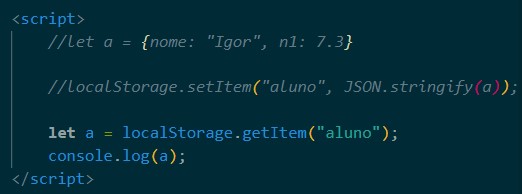
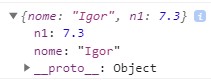
➜ **OUTROS TIPOS DE DADOS:**  Vamos ver se conseguimos armazenar outros tipos de dados:



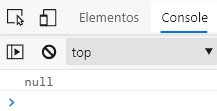
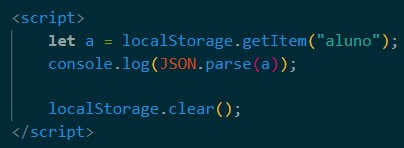
Mas esse número está preto! e como já vimos, o número não fica preto, o que costuma ser preto é string!



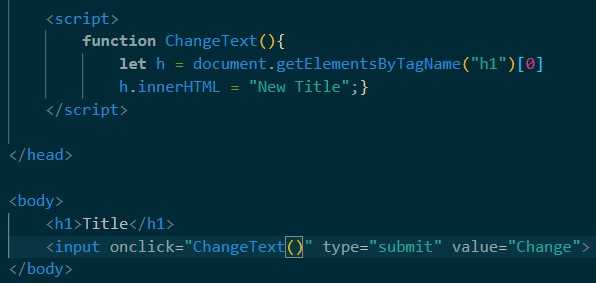
Ou seja, o JS transforma o nosso número em string e depois armazena ela no localStorage, então o localStorage só vai poder armazenar STRINGS! Então para poder armazenar outros tipos de dados sem ser string você precisa fazer uso do JSON junto com localStorage:

1. GRAVEI!
2.  Assim ele lê como STRING!
3.  E assim ele lê como um OBJETO JS, reconhecendo inclusive os números!

➜ **localStorage.clear:**  Vai apagar TUDO que está na memória!!!



➤**TIMER E INTERVALO:** Vamos supor que:

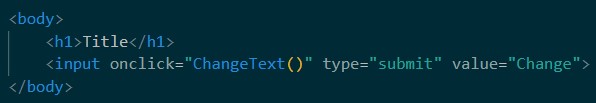


Quando você clica em mudar, o novo título aparece!

➜ **“TIMER”:**  É quando você quer executar uma função, só que você quer que tenha um delay entre o disparo dessa função, alguma coisa que inicie a execução daquela função, e a função ser executada de fato. Estou falando função, mas poderia ser QUALQUER código.

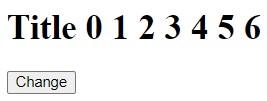
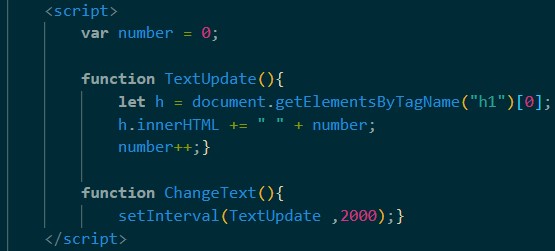
Existe um método no window chamado: “setTimeout”. O setTimeout recebe dois argumentos, o primeiro é o que você quer fazer (normalmente uma função), e o segundo argumento é o tempo de delay que você quer, contabilizado em milisegundos (1 segundo = 1000 milisegundos).

Obs: como vimos anteriormente, o window pode ser ocultado.



 Quando clica: 2 segundos depois:

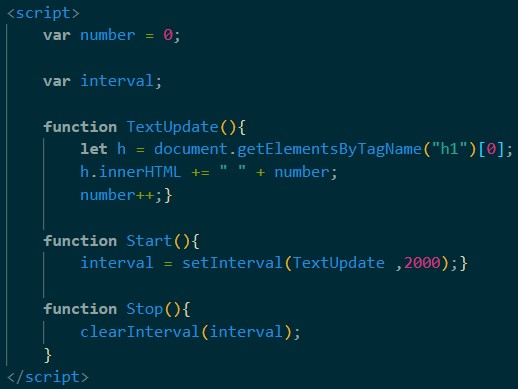
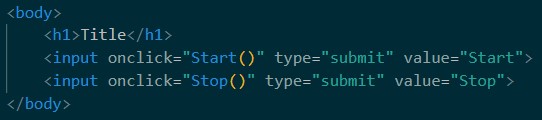
➜ **“INTERVAL”:**  Se quisermos que a execução seja repetidamente de acordo com o intervalo de tempo de uma execução para outra, aí utilizaremos o “setInterval”.

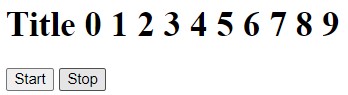
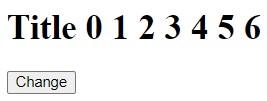


Quando clicar em “change” vai ir adicionando um número a cada 2 segundos, infinitamente!!!

E SE EU QUISER PARAR? Não quiser que a contagem continue eternamente?

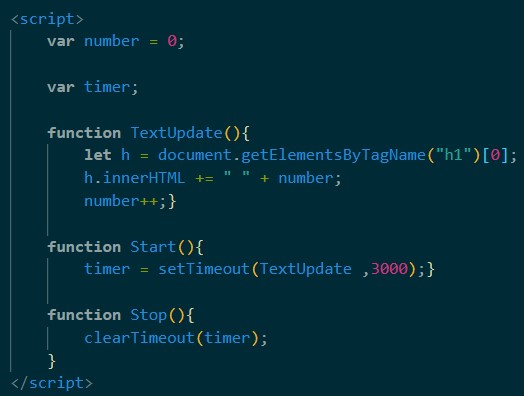
➜ **“CLEAR INTERVAL”:**  É a função que faz o intervalo parar:



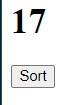
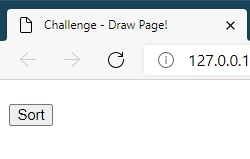


Você pode mandar parar e iniciar quantas vezes quiser, quando inicia novamente, volta de onde tinha parado.

➜ **“CLEAR Timeout”:**  Mesma coisa do outro, só que para o “Timeout”, mas como o “Timeout” só contabiliza uma única vez, o Clear Timeout só serve para parar a contagem (por algum motivo) ANTES de contabilizar esse único elemento.



➤**DESAFIO - CRIANDO UMA PÁGINA DE SORTEIO:** Vamos lá:



Fazendo diferentemente, com um Array de nomes:

Problemas para ter em mente:

1. A contagem dos códigos começa do 0, então como são 5 pessoas, vai de 0 até 4.
2. O random vai de 0 até 0,99999…, então como vai multiplicar por 5 pessoas, vai de 0 até 4,9999…, para arredondar para cima, em vez de usar “Math.round”, vamos usar “Math.floor”

